AMENDMENTS TO THE CLAIMS

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1. (Previously presented) A method for selectively monitoring store instructions to support transactional execution of a process, comprising:

starting a transactional execution of a block of instructions in a program, wherein starting the transactional execution involves executing an explicit instruction implemented in hardware to start the transactional execution;

encountering a store instruction during the transactional execution, wherein changes made during the transactional execution are not committed to the architectural state of a processor until the transactional execution successfully completes;

determining whether the store instruction is a monitored store instruction or an unmonitored store instruction by analyzing the store instruction;

if the store instruction is a monitored store instruction,

performing a corresponding store operation, and

store-marking a cache line associated with the store instruction to facilitate subsequent detection of an interfering data access to the cache line from another process; and

if the store instruction is an unmonitored store instruction, performing the corresponding store operation without store-marking the cache line.

2. (Original) The method of claim 1, wherein prior to executing the program, the method further comprises generating the instructions for the program, wherein generating the instructions involves:

determining whether store operations that take place during transactional execution need to be monitored;

generating monitored store instructions for store operations that need to be monitored; and

generating unmonitored store instructions for store operations that do not need to be monitored.

3. (Original) The method of claim 2, wherein determining whether a store operation needs to be monitored can involve examining a data structure associated with the store operation to determine whether the data structure is a "protected" data structure for which stores need to be monitored, or an "unprotected" data structure for which stores do not need to be monitored.

4. (Original) The method of claim 2, wherein determining whether a store operation needs to be monitored can involve determining whether the store operation is directed to a heap, wherein stores from the heap need to be monitored and stores from outside the heap do not need to be monitored.

5. (Original) The method of claim 2, wherein determining whether a store operation needs to be monitored can involve allowing a programmer to determine if the store operation needs to be monitored.

6. (Previously presented) The method of claim 1, wherein determining whether the store instruction is a monitored store instruction involves examining an op code of the store instruction.

7. (Previously presented) The method of claim 1, wherein determining whether the store instruction is a monitored store instruction involves examining an address associated with the store instruction to determine whether the address falls within a range of addresses for which stores are monitored.

8. (Original) The method of claim 7, wherein examining the address involves comparing the address with one or more boundary registers.

9. (Original) The method of claim 7, wherein examining the address involves examining a Translation Lookaside Buffer (TLB) entry associated with the address.

10. (Original) The method of claim 1, wherein if an interfering data access from another process is encountered during transactional execution of the block of instructions, the method further comprises:

discarding changes made during the transactional execution; and

attempting to re-execute the block of instructions.

11. (Original) The method of claim 1, wherein if transactional execution of the block of instructions completes without encountering an interfering data access from another process, the method further comprises:

committing changes made during the transactional execution to the architectural state of the processor; and

resuming normal non-transactional execution of the program past the block of instructions.

12. (Original) The method of claim 1, wherein an interfering data access can include:

a store by another process to a cache line that has been load-marked by the process; and

a load or a store by another process to a cache line that has been store-marked by the process.

13. (Original) The method of claim 1, wherein the cache line is store-marked in the cache level closest to the processor where cache lines are coherent.

14. (Original) The method of claim 1, wherein a store-marked cache line indicates at least one of the following:

loads from other processes to the cache line should be monitored;

stores from other processes to the cache line should be monitored; and

stores to the cache line should be buffered until the transactional execution completes.

15. (Previously presented) An apparatus that selectively monitors store instructions to support transactional execution of a process, comprising:

a start transactional execution mechanism configured to start a transactional execution of a block of instructions in a program, wherein starting the transactional execution involves executing an explicit instruction implemented in hardware to start the transactional execution;

an execution mechanism within a processor;

wherein the execution mechanism is configured to support the transactional execution, and wherein changes made during the transactional execution are not committed to the architectural state of a processor until the transactional execution successfully completes;

wherein upon encountering a store instruction during transactional execution, the execution mechanism is configured to, determine whether the store instruction is a monitored store instruction or an unmonitored store instruction by analyzing the store instruction,

if the store instruction is a monitored store instruction, to perform a corresponding store operation, and to store-mark a cache line associated with the store instruction to facilitate subsequent detection of an interfering data access to the cache line from another process; and

if the store instruction is an unmonitored store instruction, to perform the corresponding store operation without store-marking the cache line.

16. (Original) The apparatus of claim 15, further comprising an instruction generation mechanism configured to:

determine whether store operations that take place during transactional execution need to be monitored;

generate monitored store instructions for store operations that need to be monitored; and to

generate unmonitored store instructions for store operations that do not need to be monitored.

17. (Original) The apparatus of claim 16, wherein the instruction generation mechanism is configured to determine whether a store operation needs to be monitored by examining a data structure associated with the store operation to determine whether the data structure is a "protected" data structure for which stores need to be monitored, or an "unprotected" data structure for which stores do not need to be monitored.

18. (Original) The apparatus of claim 16, wherein the instruction generation mechanism is configured to determine whether a store operation needs to be monitored by determining whether the store operation is directed to a heap, wherein stores from the heap need to be monitored and stores from outside the heap do not need to be monitored.

19. (Original) The apparatus of claim 16, wherein the instruction generation mechanism is configured to determine whether a store operation needs to be monitored by allowing a programmer to determine if the store operation needs to be monitored.

20. (Original) The apparatus of claim 15, wherein the execution mechanism is configured to determine whether a store operation needs to be monitored by examining an op code of the store instruction.

21. (Original) The apparatus of claim 15, wherein the execution mechanism is configured to determine whether a store operation needs to be monitored by examining an address associated with the store instruction to determine whether the address falls within a range of addresses for which stores are monitored.

22. (Original) The apparatus of claim 21, wherein the execution mechanism is configured to examine the address by comparing the address with one or more boundary registers.

23. (Original) The apparatus of claim 21, wherein the execution mechanism is configured to examine the address by examining a Translation Lookaside Buffer (TLB) entry associated with the address.

24. (Original) The apparatus of claim 15, wherein if an interfering data access from another process is encountered during transactional execution of the block of instructions, the execution mechanism is configured to:

discard changes made during the transactional execution; and to

attempt to re-execute the block of instructions.

25. (Original) The apparatus of claim 15, wherein if transactional execution of the block of instructions completes without encountering an interfering data access from another process, the execution mechanism is configured to:

commit changes made during the transactional execution to the architectural state of the processor; and to

resume normal non-transactional execution of the program past the block of instructions.

26. (Original) The apparatus of claim 15, wherein an interfering data access can include:

a store by another process to a cache line that has been load-marked by the process; and

a load or a store by another process to a cache line that has been store-marked by the process.

27. (Original) The apparatus of claim 15, wherein the cache line is store-marked in the cache level closest to the processor where cache lines are coherent.

28. (Original) The apparatus of claim 15, wherein a store-marked cache line indicates at least one of the following:

loads from other processes to the cache line should be monitored;

stores from other processes to the cache line should be monitored; and

stores to the cache line should be buffered until the transactional execution completes.

29. (Previously presented) A computer system that selectively monitors store instructions to support transactional execution of a process, comprising:

a processor;

a memory;

a start transactional execution mechanism within the processor configured to start a transactional execution of a block of instructions in a program, wherein starting the transactional execution involves executing an explicit instruction implemented in hardware to start the transactional execution;

an execution mechanism within the processor;

wherein the execution mechanism is configured to support the transactional execution, wherein changes made during the transactional execution are not committed to the architectural state of a processor until the transactional execution successfully completes;

wherein upon encountering a store instruction during transactional execution, the execution mechanism is configured to,

determine whether the store instruction is a monitored store instruction or an unmonitored store instruction by analyzing the store instruction,

if the store instruction is a monitored store instruction, to perform a corresponding store operation, and to store-mark a cache line associated with the store instruction to facilitate subsequent detection of an interfering data access to the cache line from another process; and if the store instruction is an unmonitored store instruction, to perform the corresponding store operation without store-marking the cache line.